

Cloud computing: google firebase firestore optimization analysis

Andi Bahtiar Semma¹, Mukti Ali¹, Muh Saerozi¹, Mansur¹, Kusrini²

¹Department of Communication and Broadcasting, Faculty of Dakwah, Universitas Islam Negeri Salatiga, Salatiga, Indonesia

²Department of Informatics Engineering, Postgraduate Programme, Universitas AMIKOM Yogyakarta, Yogyakarta, Indonesia

Article Info

Article history:

Received Sep 14, 2022

Revised Nov 2, 2022

Accepted Nov 7, 2022

Keywords:

Cloud computing

Cost analysis

Data structure optimization

Firebase

Firestore

ABSTRACT

Cloud computing is a new paradigm that provides end users with a secure, personalized, dynamic computing environment with guaranteed service quality. One popular solution is Google cloud firestore, a global-scale not only structured query language (NoSQL) document database for mobile and web apps. Recent research on cloud-based NoSQL databases often discusses the difference between them and SQL databases and their performance. However, using cloud-based NoSQL databases such as firestore is tricky without any scientific comparison methodology, and it needs analysis of how its particular systems work. This study aims to discover what is the best design that could be implemented to optimize data read cost, response size, and time regarding the cloud firestore database. In this study, we develop a grade point average (GPA)-report mocking application to assess data read based on our institution's needs. This application consists of three functions. Add the graduated GPA and students' names, and view the ten highest GPAs, GPA average, and total graduated students. The finding indicates that aggregating data on the client side or utilizing the Google cloud function trigger, then updating aggregation data in one transaction significantly reduces document read count (cost), response size, and time.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Andi Bahtiar Semma

Department of Communication and Broadcasting, Faculty of Dakwah, Universitas Islam Negeri Salatiga
50716 Salatiga, Central Java, Indonesia

Email: andisemma@uinsalatiga.ac.id

1. INTRODUCTION

Cloud computing rises as a new paradigm that provides developers with a reliable, personalized dynamic computing environment with guaranteed quality of service [1], [2]. The national institute of standards and technology (NIST) defines cloud computing as a model for enabling ubiquitous, convenient, on-demand network access to a configurable pool of shared computing resources that can be rapidly provisioned and released with low management effort. Minimal management or interaction with a service provider. This cloud model includes five essential features, three service models, and four deployment models, as illustrated in Figure 1 [3]. Cloud computing makes use of a number of service models, including infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), software-as-a-service (SaaS), and many others [4]. The essential advantage of cloud computing is consolidated data; having all the data in one location helps with forensic preparation, leading to faster, more coordinated incident response. IaaS providers can create a specialized forensic server within the cloud with centralized data ready to utilize when needed [5]. When hosting conventional infrastructure and platforms on-premises, it is typically necessary to manually expand infrastructure resources like virtual machines and software development platforms [6]. Because of all the capabilities, it provides free access. Google firebase is an excellent and powerful platform available today [7].

One of the services offered by Google's firebase platform, which offers a variety of products and services for application development, is the realtime database service [8]. Firebase is a cloud-based database [9] appropriate for applications that demand frequent data updates. Google made the firebase database to aid programmers in the development of applications. Server-side coding is no longer necessary for programmers because firebase has handled it. Because firebase has dealt with the server side, programmers designing client-server applications appear to be coding as if they were writing a stand-alone program (client side) [10]. Google cloud firestore is a not only structured query language (NoSQL), document-based cloud-host database solution. Each document is organized into collections and may be linked to other subcollections. It has queries that are significantly faster and more efficient than firebase real-time database, as well as improved scalability and the speed advantage it has over firebase real-time database.

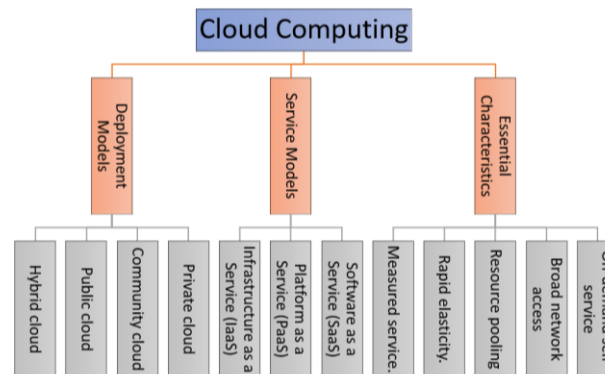


Figure 1. Cloud-computing five essential features, three service models, and four deployment models [3]

Firebase real-time database is ascribed to all queries being indexed by default, ensuring that query performance is proportional to the size of the result set data, as opposed to structured query language (SQL) database, whose querying speed diminishes as data expands [11]. The cloud firestore includes a NoSQL, document-oriented database, with data saved in JavaScript object notation (JSON) format [12]. NoSQL databases were created to address the issues of high volume, multi-source, and multi-format data processing in big data situations. NoSQL databases are a non-relational, horizontally scalable database management system that runs on standard-configuration machines [13]. Based on various data models, NoSQL data stores are highly flexible, scalable, and effective data management platforms for big data [14]. JSON document storage, along with a simple NoSQL style application programming interface (API)s, enables a lightweight, flexible development strategy for relational data, in contrast to the traditional schema-rigid SQL approach. In these operational stores, collections of schema-flexible document entities can be created, read, updated, and deleted (CRUD). On the other hand, traditional relational databases offer comparable functions but on organized rows in a table [15].

In many recent studies, there have been many discussions on comparing SQL with NoSQL database performance [10], [16]–[21]. Implement SQL and NoSQL databases in mobile applications, analyzing BIG data's type, pros, cons, characteristics, and features on SQL and NoSQL databases [22]–[24]. However, using cloud-based NoSQL databases such as firestore is tricky without any scientific comparison methodology, and it needs analysis of how its particular systems work. This study aims to provide the best scenario when using Firebase cloud firestore (NoSQL database) in real-world applications and how data structure affects the bill, response time, and size to get the most efficient and effective method.

2. METHOD

This study uses firebase cloud firestore as a database and uses Google admin software development kit (SDK) version 9.9.2 to interact with it. To achieve the goals of this study, we use general experiments and primary mean, median, and statistical mode methods. We use 'Faker' as dummy data to generate a 100 grade point average (GPA)s, student identifications (ID), and name data. We developed a simple GPA-report mocking application to assess data read to replicate real-world applications based on our institution's needs. This application consists of 3 simple functions. Add the graduated GPA, student ID, and name, and view the ten highest GPAs, GPA average, and total graduated students. This study consists of four analyses: i) Cost

related factors that affect the bill [25]; ii) Document read count; iii) Payload Size; and; iv) response time. We write a function to count every document's response for document read count. As for payload size and response time, we use chrome dev tools and [26] hyper text transfer protocol (HTTP) archive (HAR) to comma-separated values (CSV) online tool to convert its report to analyze it in a spreadsheet document.

When using cloud Firestore, some factors affect the bill. The number of documents that have been read, written, and deleted. The amount of storage used by the database, including metadata and index overhead, and the amount of network bandwidth used [27]. The price varies by location; we utilized the Jakarta region as an example in this study. The units of measurement for storage and bandwidth are gigabytes (GiB).

$$1 \text{ GiB} = 2^{30} \text{ bytes} \quad (1)$$

The daily read price is used:

$$\frac{\text{document read rate} \times (\text{daily document reads} - 50.000)}{100.000} \quad (2)$$

the base price for document reading is counted per 100,000 and 50,000 free reads daily. The written daily price is used:

$$\frac{\text{document write rate} \times (\text{daily document writes} - 20.000)}{100.000} \quad (3)$$

the base price for document writing is counted per 100.000, and 20.000 free writes daily. The daily delete price is used:

$$\frac{\text{document delete rate} \times (\text{daily document deletes} - 20.000)}{100.000} \quad (4)$$

The base price for document deleting is counted per 100,000 and 20,000 free deletes daily. Write and delete is simply measured; each set(), update(), or delete() request is considered as one [27].

Firestore charges the amount of data that is stored and storage overhead. Storage overhead includes composite indexes, metadata, and automatic indexes. The metadata on each document includes; i) the document ID, which consists of the document's name and the collection ID; ii) each field's name and value; and iii) any document-related single-field and composite indexes. The collection ID, field values, and document names are all included in each index entry, depending on how the index is defined. The amount of data stored in Cloud Firestore is calculated monthly and applied as shown in (5).

$$(\text{data stored} - 1\text{GB}) \times \text{stored document rate} \quad (5)$$

The response size, the cloud firestore database's location, and the response's destination all influence the cost of network bandwidth for a cloud firestore request. Protocol overhead, such as secure sockets layer (SSL), is not considered when calculating bandwidth use on the network. Requests for cloud security rules for Firestore are not measured against bandwidth use on the network. Ingress and egress within an area are unrestricted, as are ingress and egress across regions within a multi-region. Firebase only charges for Google cloud queries across regions; it does not charge for regional traffic in the United States or requests from sources other than Google cloud. The first ten GiBs are given free per month.

Five factors affect the cost of cloud firestore: i) reads, writes, and deletes; ii) listens to query results; iii) cloud firestore security rules; iv) storage size and; and v) network bandwidth. Some factors, like listening to query results and Security rules, are considered static variables because they stay the same regardless of what scenario is used. We assume the storage size and network bandwidth is similar to any scenario in this study because the data center we choose is similar to our targeted users.

3. RESULTS AND DISCUSSION

3.1. Scenario 1

The standard data model for no relational data with large entries is to write every data as a single document. The data set used in this scenario is students' GPA data consisting of name and GPA. We need a sorting function to determine what is the ten highest GPAs. Firestore has a built-in sorting feature to get it done. We use orderBy() and limit(); by using this feature, we reduce from millions of reads to only the data we want, i.e., ten. This scenario has some limitations: i) an orderBy() clause checks for the existence of the specified fields as well; ii) when a filter with a range comparison (=, >, >=) is included, the first ordering must

be on the same field; iii) cannot order a query by any field included in (=) equality or in a clause. Additionally, we could swiftly locate documents in extensive collections using cloud firestore's advanced queries.

```

“studentsGPAs”:
  ” 12digitsUID” {
    name: ‘string’,
    ID: ‘string’
    GPA: integer,
  }

```

(6)

3.1.1. Document read count

To count total students and average GPA, we read all the data, send it to the client side and do the aggregation on the client side. The document read count is the same as the number of documents fetched. In this scenario, two server calls are made to fulfill functional requirements. First, get all data, and second, get the ten highest GPA data. This server call model is illustrated in Figure 2.

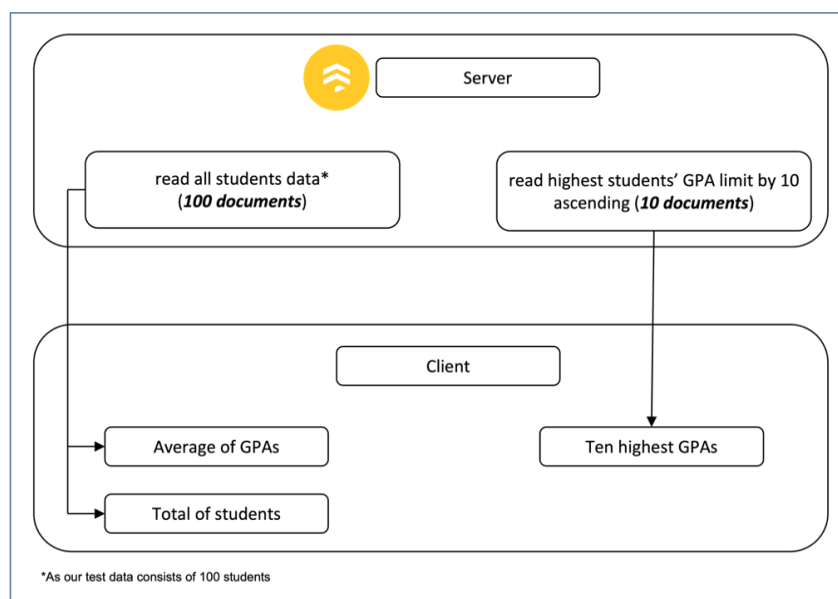


Figure 2. The illustration of scenario one

We can reduce document read count by only making one server call (read all students' data) and then aggregate and sort data on the clients' side. With this approach, the document's read count is the same as the student's data count. Nothing wrong with the data scheme here. The problem arises when numerous documents are read and sent to the client. Assume that there are hundreds or even millions or billions of documents. It will mess up here, as the document read and network bandwidth will be massive. Let us assume we have 20 thousand students and a thousand active users daily, and every user only makes one server call. The size of all documents will be:

$$20.000 \times 1000 \times 1 = 20 \text{ Millions document reads} \quad (7)$$

with 20 million reads per day, the cost will be:

$$\frac{(20.000.000 - 50.000) \times \$ 0.038}{100.000} = \$ 7.581 \quad (8)$$

3.1.2. Response size

The response size of 100 students' test data is 8.6 Kb, as shown in Figure 3. Response size will grow as the students' data grows. Big response size will affect compute time and user experience. Let us assume we have about 20 thousand students, and the response size will be:

$$8.8Kb \text{ (per 100 students)} \times 200 = 1.6 MB \tag{9}$$

Name	Status	Initiator	Size
<input checked="" type="checkbox"/> ?colName=studentsGP...	200	App.vue?11c4:211	8.8 kB
<input type="checkbox"/> ?colName=studentsGP...	200	App.vue?11c4:211	8.8 kB
<input type="checkbox"/> ?colName=studentsGP...	200	App.vue?11c4:211	8.8 kB
<input type="checkbox"/> ?colName=studentsGP...	200	App.vue?11c4:211	8.8 kB
<input type="checkbox"/> ?colName=studentsGP...	200	App.vue?11c4:211	8.8 kB
<input type="checkbox"/> ?colName=studentsGP...	200	App.vue?11c4:211	8.8 kB
<input type="checkbox"/> ?colName=studentsGP...	200	App.vue?11c4:211	8.8 kB
<input type="checkbox"/> ?colName=studentsGP...	200	App.vue?11c4:211	8.8 kB

Figure 3. Response size of scenario 1

3.1.3. Response time

We conducted a hundred server calls in this scenario to get network data. Before we analyze the data, we make a histogram, as shown in Figure 4, to remove any outliers from the data to get more reliable data. All presented data is in milliseconds. In this scenario, we get a minimum response time of 83 milliseconds, a maximum of 144 milliseconds, a mean of 99 milliseconds, a Median of 97, a Mode of 98 milliseconds, and a Standard Deviation of 9 milliseconds. Detailed information is shown in Table 1.

Table 1. Response time data-scenario 1

Analysis	Response time in milliseconds
Min	83
Max	144
Mean	99
Median	97
Mode	98
Standard deviation	9

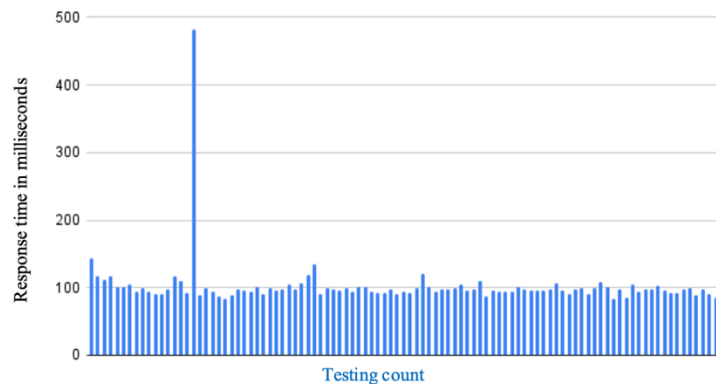


Figure 4. Response time data histogram-scenario 1

3.2. Scenario 2

In the second scenario, we made a summary document that stores the total students count, average GPA, and ten highest GPAs. First, create a server call to request compilation data, then update the summary documents with new student data added. This function can be done with two models. The first model is Client-side transactions, which means that clients request a summary document and then add new students' data to the summary document requested before, then post the summarized new data and add new students' data in one transaction. That means if one of the functions fails, either updating the summary or adding new students' data, all the requests will be canceled. The illustrations of this model can be seen in Figure 5. Each time a new student's data is added to the collection. These aggregations must be updated to keep them consistent. Performing the add and update in a single transaction is one technique to achieve consistency. The aggregate data is consistent with the underlying collection by using a transaction. This solution has some limitations: i) security-client-side transactions require authorizing the client to update the database's aggregate data. While

enhanced security controls can help mitigate this strategy's risks, it may not be acceptable in all instances; ii) offline support-when the user's device is offline, client-side transactions will fail. The developer will need to address this in their app and retry at the right moment; iii) performance-multiple requests to the cloud firestore backend may be required if the transaction involves multiple read, write, and update activities. Because this could take a long time on a mobile device; and iv) write rates-because cloud firestore documents can only be updated once every second, this method may not be suitable for regularly updated aggregations. A transaction also retries a certain number of times before failing.

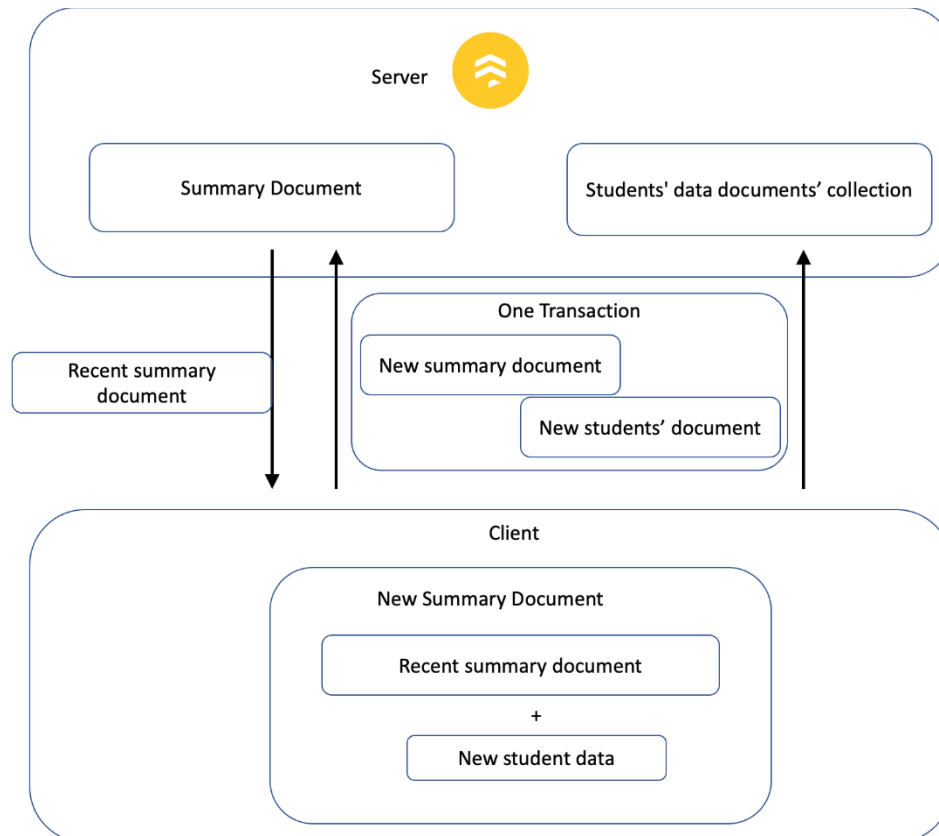


Figure 5. Client-side aggregation

The second model uses Firebase cloud function if client-side transactions are not applicable. To perform this model, we update the summary data using a cloud function each time a new entry is added to the students' data collection. This method transfers the work from the client to a hosted function, allowing the application to add points without waiting for a transaction to finish. We no longer need to allow clients to write access to the aggregate data because code executed in a Cloud Function is not restricted by security constraints. The Illustration can be seen in Figure 6.

This model has limitations: i) cost-each new student's data post will trigger a cloud function call, potentially increasing prices.; ii) latency-because the aggregation process has been offloaded to a cloud function, the application will not view updated data until the cloud function has been completed. This function may take longer than completing the transaction locally, depending on the speed of the cloud function; iii) write rates-because cloud firestore documents can only be updated once every second, this method may not be suitable for regularly updated aggregations. If a transaction tries a finite number of times to read a document that has been updated outside of the transaction, it will fail.

3.2.1. Document read counts

To get the total students' average GPA and the ten highest GPA data, it only performs one server call and reads only one document. The document read count is the same as the number of documents fetched, which is the summary document. Let us assume we have 20 thousand students and a thousand active users daily, and every user only makes one server call per visit, and the size of all documents will be:

$$1 \text{ document} \times 1000 = 1.000 \text{ document reads} \tag{10}$$

With 1,000 documents reads per day, the cost will be \$0 because it is still under the free daily quota of 50,000 free documents read per day.

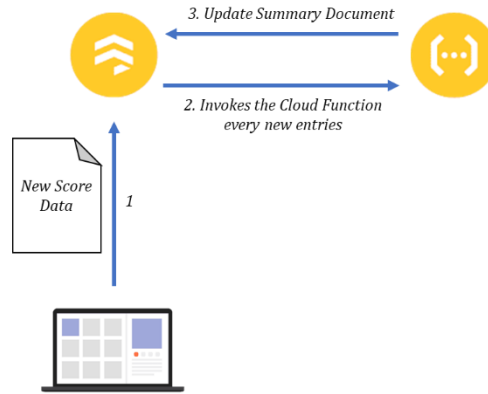


Figure 6. Cloud function invocation

3.2.2. Response size

The response size of 100 students’ summary test data is 964 bytes, as shown in Figure 7. Response size will not grow as the students’ data grow because it only contains summarized data regardless of how many students data. This scenario perform better than the previous one.

Name	Status	Initiator	Size	▲
?colName=summary&id...	200	App.vue?11c4:56	964 B	
?colName=summary&id...	200	App.vue?11c4:56	964 B	
?colName=summary&id...	200	App.vue?11c4:56	964 B	
?colName=summary&id...	200	App.vue?11c4:56	964 B	
?colName=summary&id...	200	App.vue?11c4:56	964 B	
?colName=summary&id...	200	App.vue?11c4:56	964 B	
?colName=summary&id...	200	App.vue?11c4:56	964 B	
?colName=summary&id...	200	App.vue?11c4:56	964 B	
?colName=summary&id...	200	App.vue?11c4:56	964 B	
?colName=summary&id...	200	App.vue?11c4:56	964 B	
?colName=summary&id...	200	App.vue?11c4:56	964 B	
?colName=summary&id...	200	App.vue?11c4:56	964 B	
?colName=summary&id...	200	App.vue?11c4:56	964 B	
?colName=summary&id...	200	App.vue?11c4:56	964 B	
?colName=summary&id...	200	App.vue?11c4:56	964 B	

Figure 7. Response size of scenario 2

3.2.3. Response time

We conducted a hundred server calls in this scenario to get network data. Before we analyze the data, we make a data histogram, as shown in Figure 8. Remove any outliers from the data so we can get more reliable data. All presented data is in milliseconds. In this scenario, we get a minimum response time of 61 milliseconds, maksimum of 92 milliseconds, mean of 71 milliseconds, median of 71 milliseconds, Mode of 69 milliseconds, and standard deviation of 6 milliseconds. Detailed information is shown in Table 2.

Table 2. Response time data-scenario 2

Analysis	Response time in milliseconds
Min	61
Max	92
Mean	71
Median	71
Mode	69
Standard deviation	6

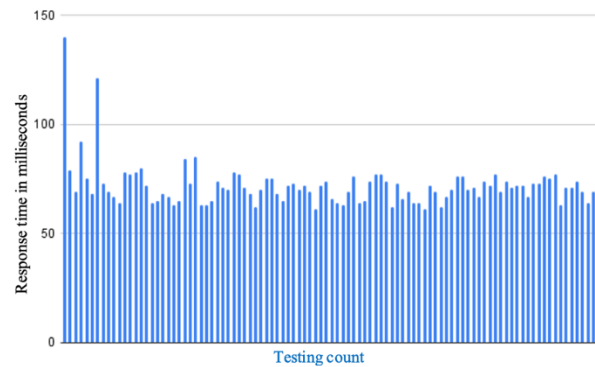


Figure 8. Response time data histogram-scenario 2

3.3. Comparison overview

In this section, we make a comparison table so it might be easier to see the differences between the two scenarios. The comparison overview is shown in Table 3. From the Table 3, scenario 2 performs better than scenario 1 in all areas. This finding is similar to the proposed NoSql data modeling by [28]–[30]. The finding and suggestions from [31], [32] also related to this study's finding that a specific NoSql database requires detailed analysis for the performance and consistent access operations. Calculating aggregation data on the client or cloud function before sending it to the database performs better in cost efficiency, response time, and response size.

Table 3. Comparison overview

Analysis	Scenario 1	Scenario 2
Daily cost	\$ 7.581	\$ 0
Document reads	100	1
Response Size	8.844 Bytes	964 Bytes
Min response time	83 ms	61 ms
Max response time	144 ms	92 ms
Mean	99 ms	71 ms
Median	97 ms	71 ms
Mode	98 ms	69 ms
Standard deviation	9 ms	6 ms

4. CONCLUSION

Using cloud-based NoSQL databases is tricky and needs analysis of how its particular systems work. This study aims to provide the best scenario when using firebase coud firestore (NoSQL database) in a real-world application. The best model for using cloud firestore in this study is to separate the summary or aggregated data on another collection and document. Calculating aggregation data on the client or cloud function before sending it to the database performs better in cost efficiency, response time, and response size. This study contributed to the best practice of using cloud firestore with specific techniques. This paper also has practical implications to help application developers newly into the NoSQL world from an SQL database background, so they can implement the best practice of using cloud firestore NoSQL database. This paper has limitations that are only analyzed on cloud firestore and use a simple data structure. Nevertheless, this paper can be the starting point for new application developers into cloud-based NoSQL databases, especially cloud firestore, to familiarize themselves with its system or concept. Future studies should attempt to use more complex data structures and different cloud-based database providers.

ACKNOWLEDGEMENTS




The Ministry of Religion of Indonesia funds this research.

REFERENCES




- [1] A. Rashid and A. Chaturvedi, "Cloud Computing Characteristics and Services A Brief Review," *International Journal of Computer Sciences and Engineering*, vol. 7, no. 2, pp. 421–426, Feb. 2019, doi: 10.26438/ijcse/v7i2.421426.

- [2] B. Alouffi, M. Hasnain, A. Alharbi, W. Alosaimi, H. Alyami, and M. Ayaz, "A Systematic Literature Review on Cloud Computing Security: Threats and Mitigation Strategies," *IEEE Access*, vol. 9, pp. 57792–57807, 2021, doi: 10.1109/ACCESS.2021.3073203.
- [3] C. Miyachi, "What is 'Cloud'? It is time to update the NIST definition?," *IEEE Cloud Computing*, vol. 5, no. 3, pp. 6–11, May 2018, doi: 10.1109/MCC.2018.032591611.
- [4] S. Heuchert, B. P. Rimal, M. Reisslein, and Y. Wang, "Design of a small-scale and failure-resistant IaaS cloud using OpenStack," *Applied Computing and Informatics*, Sep. 2021, doi: 10.1108/ACI-04-2021-0094.
- [5] V. Prakash, A. Williams, L. Garg, C. Savaglio, and S. Bawa, "Cloud and edge computing-based computer forensics: Challenges and open problems," *Electronics (Switzerland)*, vol. 10, no. 11, p. 1229, May 2021, doi: 10.3390/electronics10111229.
- [6] C. M. Mohammed and S. R. Zeebaree, "Sufficient Comparison Among Cloud Computing Services: IaaS, PaaS, and SaaS: A Review," *International Journal of Science and Business*, vol. 5, no. 2, pp. 17–30, 2021, doi: 10.5281/zenodo.4450129.
- [7] M. R. Rezoug, R. Chenni, and D. Taibi, "A New Approach for Optimizing Management of a Real Time Solar Charger Using the Firebase Platform Under Android," *Journal of Low Power Electronics and Applications*, vol. 9, no. 3, p. 23, Jul. 2019, doi: 10.3390/jlpea9030023.
- [8] N. N. Sari, M. N. Gani, R. A. Maharani Yusuf, and R. Firmando, "Telemedicine for silent hypoxia: Improving the reliability and accuracy of Max30100-based system," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 22, no. 3, pp. 1419–1426, 2021, doi: 10.11591/ijeecs.v22.i3.pp1419-1426.
- [9] P. Megantoro *et al.*, "Instrumentation system for data acquisition and monitoring of hydroponic farming using ESP32 via Google Firebase," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 27, no. 1, p. 52, Jul. 2022, doi: 10.11591/ijeecs.v27.i1.pp52-61.
- [10] I. K. G. Sudiarta, I. N. E. Indrayana, I. W. Suasnawa, S. A. Asri, and P. W. Sunu, "Data Structure Comparison Between MySQL Relational Database and Firebase Database NoSql on Mobile Based Tourist Tracking Application," *Journal of Physics: Conference Series*, vol. 1569, no. 3, p. 032092, Jul. 2020, doi: 10.1088/1742-6596/1569/3/032092.
- [11] F. M. Dahunsi, A. J. Joseph, O. A. Sarumi, and O. O. Obe, "Database management system for mobile crowdsourcing applications," *Nigerian Journal of Technology*, vol. 40, no. 4, pp. 713–727, Oct. 2021, doi: 10.4314/njt.v40i4.18.
- [12] R. Mallik, A. P. Hazarika, S. Ghosh Dastidar, D. Sing, and R. Bandyopadhyay, "Development of An Android Application for Viewing Covid-19 Containment Zones and Monitoring Violators Who are Trespassing into It Using Firebase and Geofencing," *Transactions of the Indian National Academy of Engineering*, vol. 5, no. 2, pp. 163–179, Jun. 2020, doi: 10.1007/s41403-020-00137-3.
- [13] M. Banane, A. Belangour, and E. H. Labriji, "RDF Data Management Systems Based on NoSQL Databases: A Comparative Study," *International Journal of Computer Trends and Technology*, vol. 58, no. 2, pp. 98–102, Apr. 2018, doi: 10.14445/22312803/IJCTT-V58P117.
- [14] P. Colombo and E. Ferrari, "Access control technologies for Big Data management systems: literature review and future trends," *Cybersecurity*, vol. 2, no. 1, p. 3, Dec. 2019, doi: 10.1186/s42400-018-0020-9.
- [15] Z. H. Liu *et al.*, "Native JSON Datatype Support: Maturing SQL and NoSQL convergence in Oracle Database," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3059–3071, Aug. 2020, doi: 10.14778/3415478.3415534.
- [16] J. Antas, R. R. Silva, and J. Bernardino, "Assessment of SQL and NoSQL Systems to Store and Mine COVID-19 Data," *Computers*, vol. 11, no. 2, p. 29, Feb. 2022, doi: 10.3390/computers11020029.
- [17] M. M. Eyada, W. Saber, M. M. El Genidy, and F. Amer, "Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments," *IEEE Access*, vol. 8, pp. 110656–110668, 2020, doi: 10.1109/ACCESS.2020.3002164.
- [18] J. A. Herrera-Ramirez, M. Treviño-Villalobos, and L. Viquez-Acuña, "Hybrid storage engine for geospatial data using NoSQL and SQL paradigms," *Revista Tecnología en Marcha*, Feb. 2021, doi: 10.18845/tm.v34i1.4822.
- [19] M. T. Gonzalez-Aparicio, M. Younas, J. Tuya, and R. Casado, "Evaluation of ACE properties of traditional SQL and NoSQL big data systems," in *Proceedings of the ACM Symposium on Applied Computing*, Apr. 2019, vol. Part F147772, pp. 1988–1995, doi: 10.1145/3297280.3297474.
- [20] L. Zhang, K. Pang, J. Xu, and B. Niu, "JSON-based control model for SQL and NoSQL data conversion in hybrid cloud database," *Journal of Cloud Computing*, vol. 11, no. 1, p. 23, Dec. 2022, doi: 10.1186/s13677-022-00302-9.
- [21] V. F. de Oliveira, M. A. de O. Pessoa, F. Junqueira, and P. E. Miyagi, "Sql and nosql databases in the context of industry 4.0," *Machines*, vol. 10, no. 1, p. 20, Dec. 2022, doi: 10.3390/machines10010020.
- [22] W. Ali, M. U. Shafique, M. A. Majeed, and A. Raza, "Comparison between SQL and NoSQL Databases and Their Relationship with Big Data Analytics," *Asian Journal of Research in Computer Science*, vol. 4, pp. 1–10, Oct. 2019, doi: 10.9734/ajrcos/2019/v4i230108.
- [23] D. Laksono, "Testing Spatial Data Deliverance in SQL and NoSQL Database Using NodeJS Fullstack Web App," in *Proceedings - 2018 4th International Conference on Science and Technology, ICST 2018*, Aug. 2018, pp. 1–5, doi: 10.1109/ICSTC.2018.8528705.
- [24] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi, and F. Ismaili, "Comparison between relational and NOSQL databases," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings*, May 2018, pp. 216–221, doi: 10.23919/MIPRO.2018.8400041.
- [25] Google Developers, "Firebase: Pricing," *Google Developers*, 2016. <https://firebase.google.com/pricing/> (accessed Aug. 19, 2022).
- [26] "HAR to CSV," *GitHub*. <https://hintdesk.github.io/networkhartocsv/input> (accessed Aug. 21, 2022).
- [27] Google, "Understand Cloud Firestore billing | Firebase," *Firebase*, 2022. <https://firebase.google.com/docs/firestore/pricing#europe> (accessed Aug. 19, 2022).
- [28] F. Abdelhedi, A. Ait Brahim, F. Atigui, and G. Zurfluh, "MDA-based approach for NoSQL databases modelling," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10440 LNCS, 2017, pp. 88–102.
- [29] J. Mali, F. Atigui, A. Azough, and N. Travers, "Modeldrivenguide: An approach for implementing nosql schemas," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12391 LNCS, 2020, pp. 141–151.
- [30] A. A. Imam *et al.*, "Dsp: Schema design for non-relational applications," *Symmetry*, vol. 12, no. 11, pp. 1–33, Oct. 2020, doi: 10.3390/sym12111799.
- [31] J. Pokorný, "Integration of Relational and NoSQL Databases," *Vietnam Journal of Computer Science*, vol. 6, no. 4, pp. 389–405, Nov. 2019, doi: 10.1142/S2196888819500210.
- [32] C. Asaad and K. Bařna, "NoSQL databases – seek for a design methodology," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11163 LNCS, 2018, pp. 25–40.




BIOGRAPHIES OF AUTHORS

Andi Bahtiar Semma, M.Kom.    is a lecturer and researcher Universitas Islam Negeri (UIN) Salatiga. Before it, he was a lead programmer at PT. Primayudha Mandirijaya (SRITEX Group) and the Division Head at PT. Sampoerna, Tbk. During his tenure at PT. Primayudha Mandirijaya (SRITEX Group), helped the company develop an enterprise resource planning system to manage company administration needs. He graduated from Universitas Indonesia and Universitas AMIKOM Yogyakarta, where majored in Computer Science. Andi is passionate about computer science, software engineering, and education technology. When he is not keeping busy with teaching and research. He can be contacted at email: andisemma@uinsalatiga.ac.id.






Dr. Mukti Ali    is a Dean of Dakwah Faculty of Universitas Islam Negeri (UIN) Salatiga, Indonesia and a Associate Professor on Islamic Communication and Broadcasting. His research interests including; islamic studies, communication studies, intercultural communication, cultural studies, ethnic studies and anthropology of communication. He can be contacted at email: muktiali@uinsalatiga.ac.id.






Prof. Dr. Muh Saerozi    is a vice Rector of Universitas Islam Negeri (UIN) Salatiga, Indonesia and is a Professor on Islamic Education. His research interests including; Islamic Studies, Education and Curriculum Studies, Educational Technology, Teaching Method, Special Education. When he is not keeping busy with teaching and research. He can be contacted at email: muhsaerozi@uinsalatiga.ac.id.



Prof. Dr. Mansur    is a Dean of Tarbiyah Faculty of Universitas Islam Negeri (UIN) Salatiga, Indonesia and a Professor on Islamic Education. His research interests including; Islamic Studies, Education and Curriculum Studies, Educational Technology, Teaching Method, Special Education. When he is not keeping busy with teaching and research. He can be contacted at email: mansur@uinsalatiga.ac.id.



Prof. Dr. Kusrini, M.Kom.    is a professor from Universitas AMIKOM Yogyakarta Indonesia. She finished her doctoral program from Universitas Gadjah Mada Yogyakarta Indonesia in 2010. She is interested in exploring many things about machine learning and other artificial intelligence field. She also loves in researching decision support system and database. She is member of the IEEE and IEEE Systems, Man, and Cybernetics Society. She can be contacted at email: kusrini@amikom.ac.id.